

Agile, CI, & Static Analysis

A case study in a unified approach to security and quality in the SDLC

Gordon Uchenick, Coverity



Development & Security Goals Are Not Aligned

Development Goals

Deliver Innovative New
Products to Market

Meet Product Requirements
and Delivery Schedule

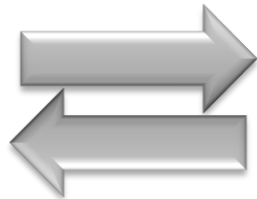
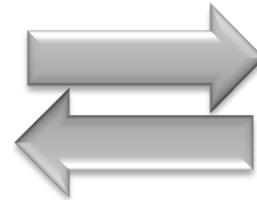
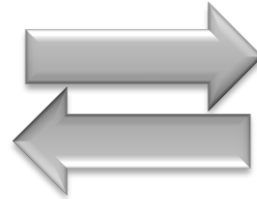
Ensure free of Quality
Defects

Security Goals

Ensure Vulnerabilities
are Not Released

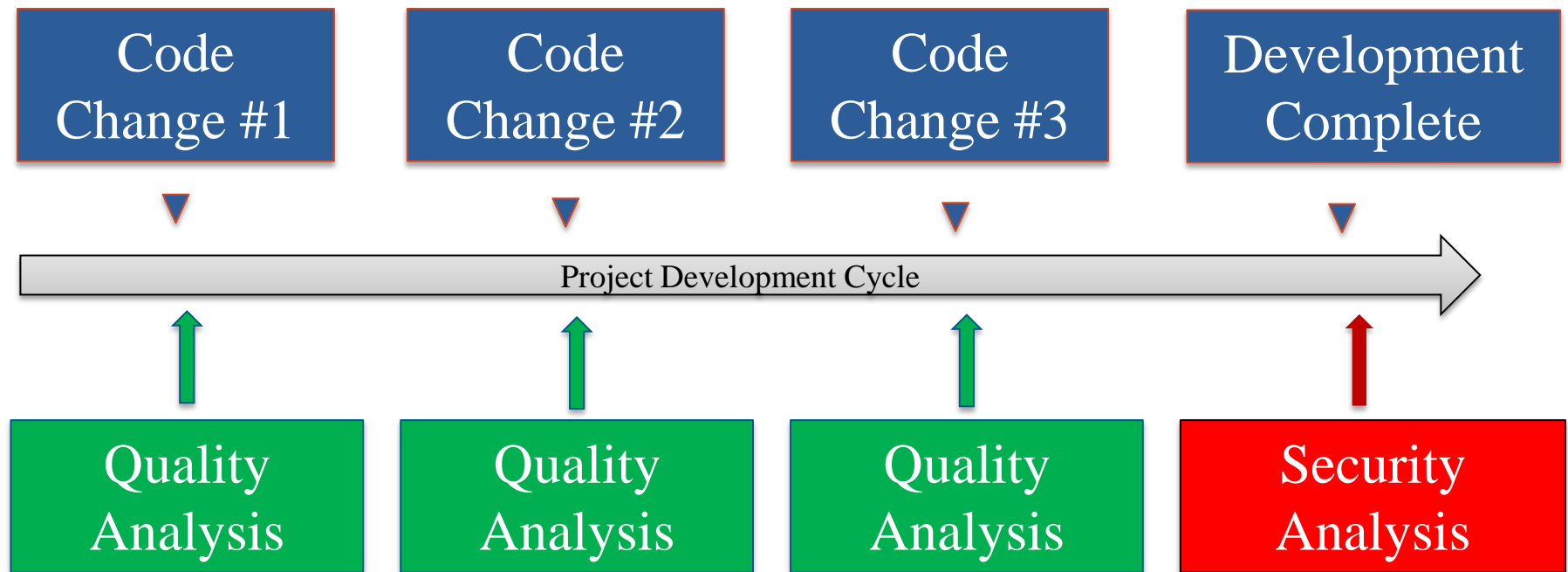
Meet Compliance and
Audit Requirements

Ensure free of
Security Vulnerabilities



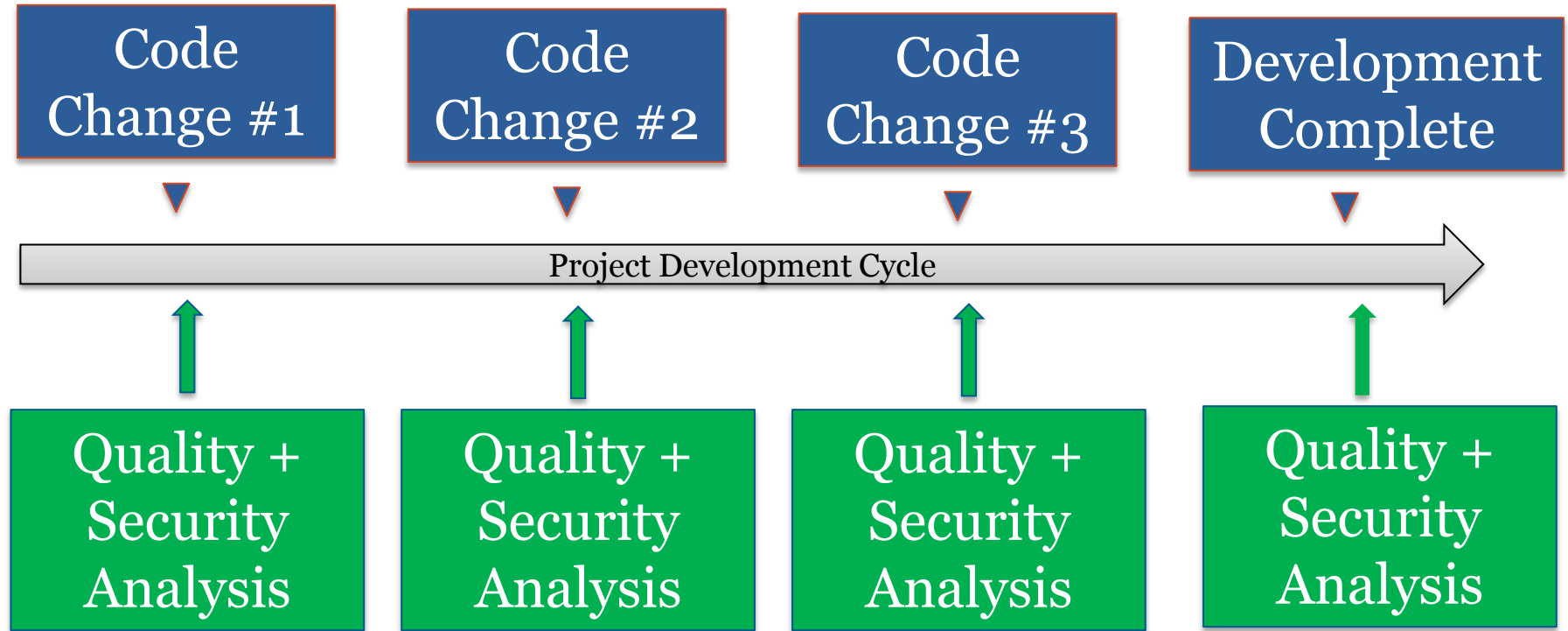
Why It's So Hard to Prevent Security Vulnerabilities?

Security Testing Happens Too Late In The Process



Increases Risk of Delays or Missing a Vulnerability

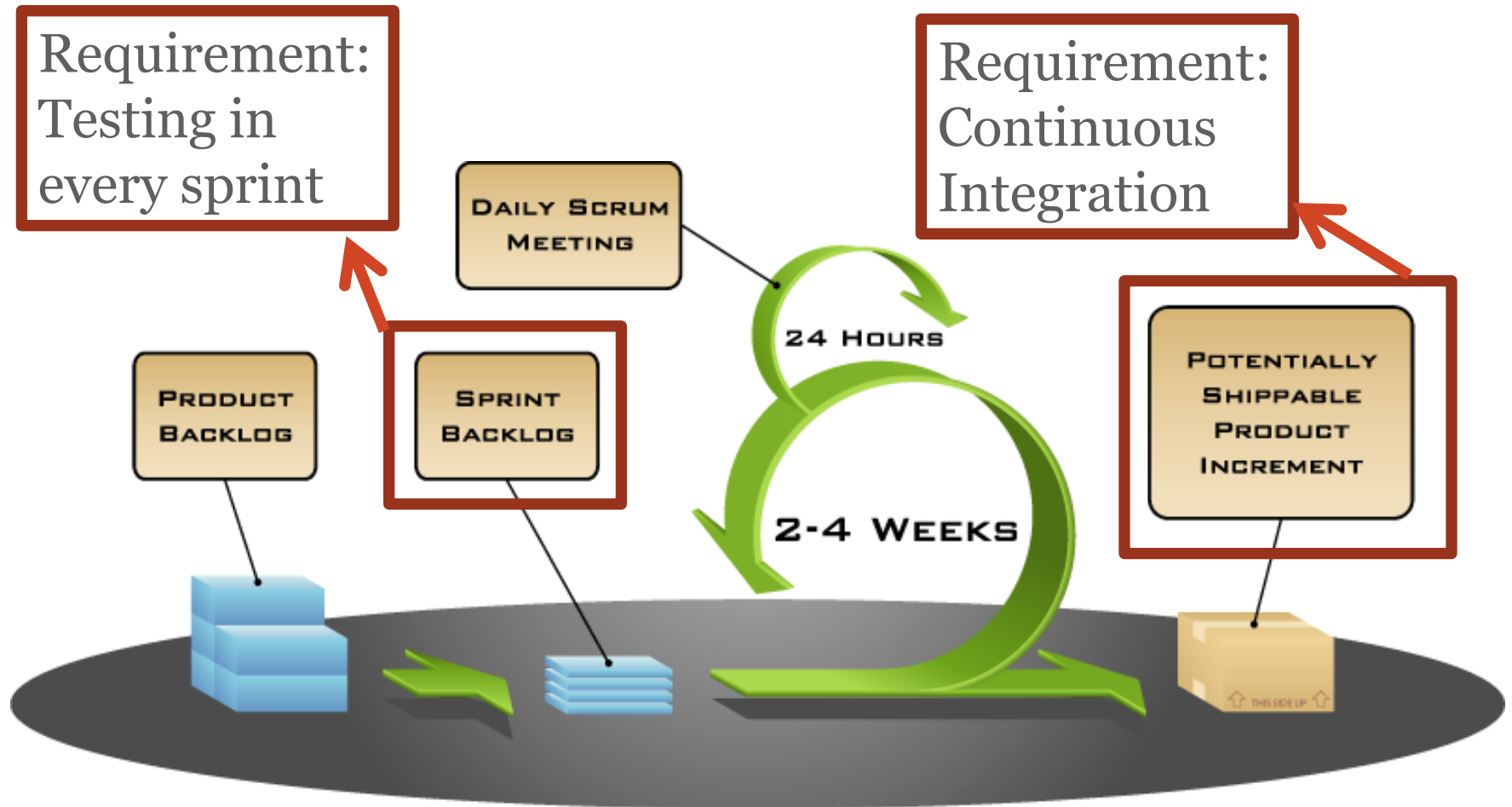
Solution – Unified Approach With Static Analysis



Quality vs Security Defects

- Is there any quality defect that is not a potential security defect?
- Remember the Rugged Software Manifesto:
 - “I recognize that my code will be used in ways I cannot anticipate, in ways it was not designed, and for longer than it was ever intended.”
 - In other words, “Every assumption I made about the user or the environment could be false.”
- Use a complete definition of security!
 - In practice, Security = Confidentiality
 - Integrity and Availability can’t be ignored
- Fix ***both*** quality (i.e., functional) and security defects at the earliest possible point in the product lifecycle.

The A(a)gile Case – New Requirements



COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE

Solution – Fast + Accurate Analysis & Integration With A(a)gile

1. Analysis for quality & security on the developers desktop
 - Clean before check-in
 - Clean before peer review
2. Analysis for quality & security defects with every build iteration
3. Analysis results in Continuous Integration workflow

Unified Quality & Security Analysis On The Developers Desktop

The screenshot displays the Eclipse IDE interface. The main editor shows a Java file named `Example.java` with a defect highlighted: `DEADLOCK defect: Two threads acquire two locks in different orders.` The defect is associated with a `static class BA implements Runnable` that uses `synchronized` blocks. Two semi-transparent boxes are overlaid on the code:

- Find and fix all defects as soon as they are introduced**
- Find, triage and fix defects in the development IDE**

On the right, the **Defect Triage** panel shows details for defect **10168: Thread deadlock**. The status is **New**, classification is **Unclassified**, severity is **Unspecified**, action is **Undecided**, and owner is **Unassigned**. The occurrences list shows two instances of the `lock_event` defect, indicating an inconsistent order of nested lock acquisitions.

At the bottom, the **Problems** panel displays a list of defects. The table below shows the first 13 defects:

CID	Checker	Status	Owner	Classification	Severity	Action	Compon...	Function	Resource	Path
10169	REVERSE_NULL	Triaged	Unassigned	Bug	Moderate	Fix Requ...	Default.O...	simple.Example.reverseN...	Example.java	...alyzer Demos\src\simple
10168	LOCK_INVERSION	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	simple.Example\$Deadloc...	Example.java	...alyzer Demos\src\simple
10167	GUARDED_BY_VIOLATION	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	philosophers.Chopstick.g...	Chopstick.java	...r Demos\src\philosophers
10166	FORWARD_NULL	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	simple.Example.forwardN...	Example.java	...alyzer Demos\src\simple
10165	FB.SWL_SLEEP_WITH_LOCK_HELD	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	philosophers.Philosopher...	Philosopher.java	...r Demos\src\philosophers
10164	FB.SIC_INNER_SHOULD_BE_STATIC	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...		OpenResources.java	...yzer Demos\src\resources
10163	FB.SE_BAD_FIELD_STORE	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...		Garden.java	...alyzer Demos\src\garden
10162	FB.FL_PUBLIC_SHOULD_BE_PROTEC...	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	resources.OpenResources...	OpenResources.java	...yzer Demos\src\resources
10161	FB.FL_PUBLIC_SHOULD_BE_PROTEC...	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	resources.OpenResources...	OpenResources.java	...yzer Demos\src\resources
10160	FB.FL_EMPTY	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	resources.OpenResources...	OpenResources.java	...yzer Demos\src\resources
10159	FB.DM_NEXTINT_VIA_NEXTDOUBLE	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	philosophers.Philosopher...	Philosopher.java	...r Demos\src\philosophers
10158	FB.DM_EXIT	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	visibility.Visibility\$2.run()	Visibility.java	...alyzer Demos\src\visibility
10157	FB.DE_MIGHT_IGNORE	New	Unassigned	Unclassified	Unspecified	Undecided	Default.O...	resources.OpenResources...	OpenResources.java	...yzer Demos\src\resources

The bottom status bar shows **Writable**, **Smart Insert**, and **79:18**.

Unified Quality & Security Analysis

The screenshot displays the Coverity Integrity Manager web interface. The top navigation bar includes links for Admin User, Sign out, Preferences, Help, About, and a search bar for CID(s). The main navigation menu shows Dashboard, Projects (selected), Configuration, and Administration. The breadcrumb trail indicates the current view is Projects > Framework > New View*.

On the left, the 'Filter results by:' section is expanded. Under 'Defect Type:', the following filters are checked and highlighted with red boxes:

- High Impact: only
- Memory - corruptions
- Memory - illegal accesses
- Resource leaks
- Uninitialized variables
- Security best practices violations

The main table displays 170 defects matching the filters. The table has columns for All, CID, Checker, Status, First Detected, File, Function, and Owner. The first few rows of the table are as follows:

All	CID	Checker	Status	First Detected	File	Function	Owner
<input type="checkbox"/>	11975	USE_AFTER_FREE	New	02/07/10	/tmp/git-1.6.0/builtin-apply.c	apply_one_fragment()	Unassigned
<input type="checkbox"/>	11971	UNINIT	New	02/07/10	/tmp/git-1.6.0/grep.c	match_one_pattern()	Unassigned
<input type="checkbox"/>	11970	UNINIT	New	02/07/10	/tmp/git-1.6.0/builtin-apply.c	check_patch()	Unassigned
<input type="checkbox"/>	11913	STRING_SIZE	New	02/07/10	/tmp/git-1.6.0/builtin-init-db.c	copy_templates()	Unassigned
<input type="checkbox"/>	11912	STRING_OVERFLOW	New	02/07/10	/tmp/git-1.6.0/merge-index.c	merge_entry()	Unassigned

Three callout boxes with arrows pointing to the interface:

- Box 1 (top): "Which defects are most critical?" points to the 'High Impact: only' filter.
- Box 2 (middle): "Which defects should I fix first?" points to the 'Memory - illegal accesses' filter.
- Box 3 (bottom): "Show me specific kinds of defects" points to the 'Security best practices violations' filter.

Continuous Integration With Jenkins

Jenkins

search



Jenkins » [DosBox](#) » [#10](#)

[ENABLE AUTO REFRESH](#)

[Back to Project](#)

[Status](#)

[Changes](#)

[Console Output](#)

[Edit Build Information](#)

[Tag this build](#)

[Coverity Defects](#)

[Previous Build](#)

[Next Build](#)



Build #10 (Aug 4, 2011 12:57:03 PM)

Success



Revision: 5
No changes.



Started by user [anonymous](#)



[147](#) matching Coverity defect(s) found.

Delete this build

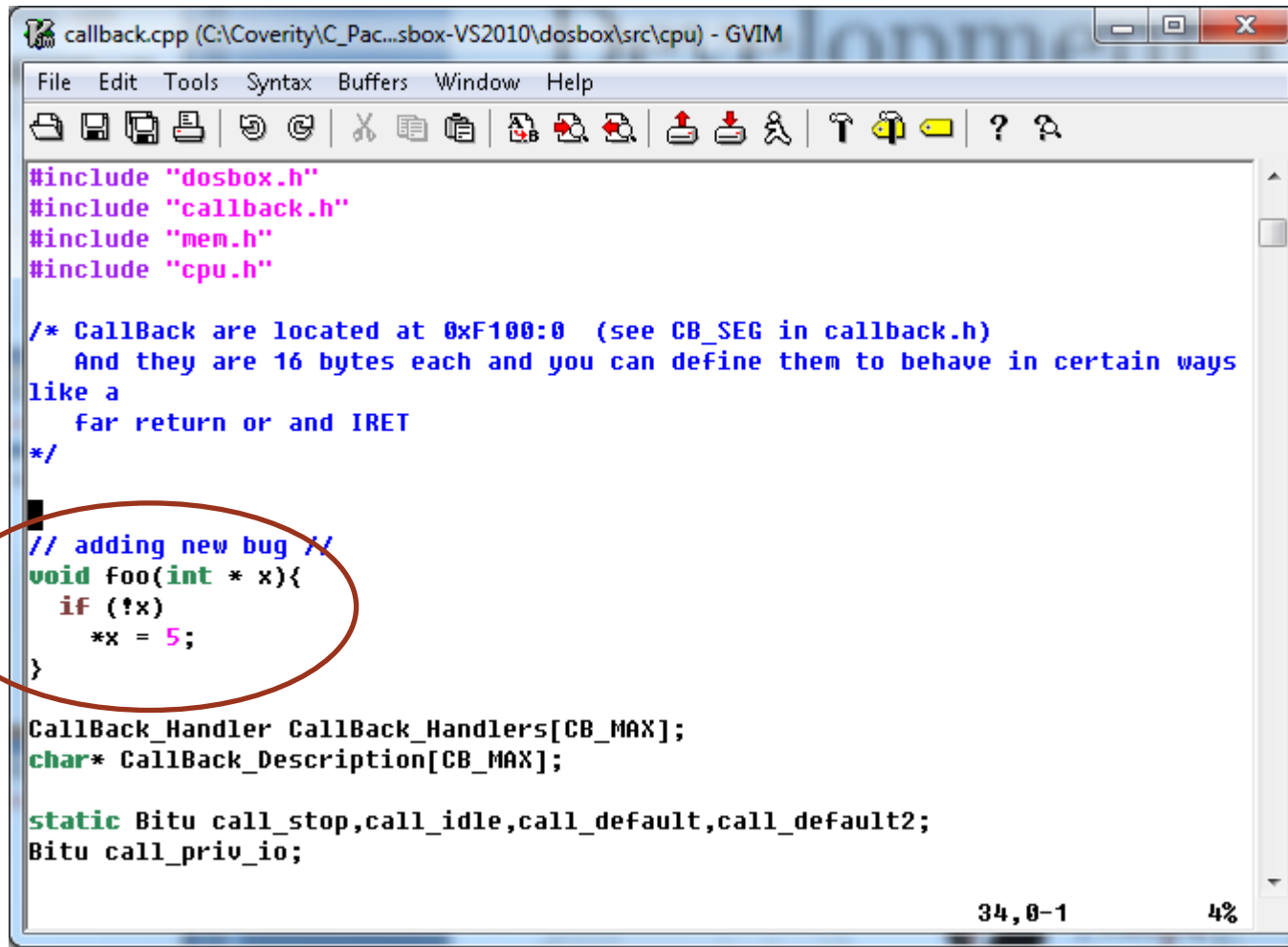
Started 1 hr 36 min ago
Took [14 min](#)

[add description](#)

A Continuous Integration build is successful only when it is free of quality & security defects

Continuous Integration Example

Jenkins Integration – agile execution



```
callback.cpp (C:\Coverity\C_Pac...sbox-VS2010\dosbox\src\cpu) - GVIM
File Edit Tools Syntax Buffers Window Help

#include "dosbox.h"
#include "callback.h"
#include "mem.h"
#include "cpu.h"

/* Callback are located at 0xF100:0 (see CB_SEG in callback.h)
   And they are 16 bytes each and you can define them to behave in certain ways
   like a
   far return or and IRET
*/

// adding new bug //
void foo(int * x){
    if (!x)
        *x = 5;
}

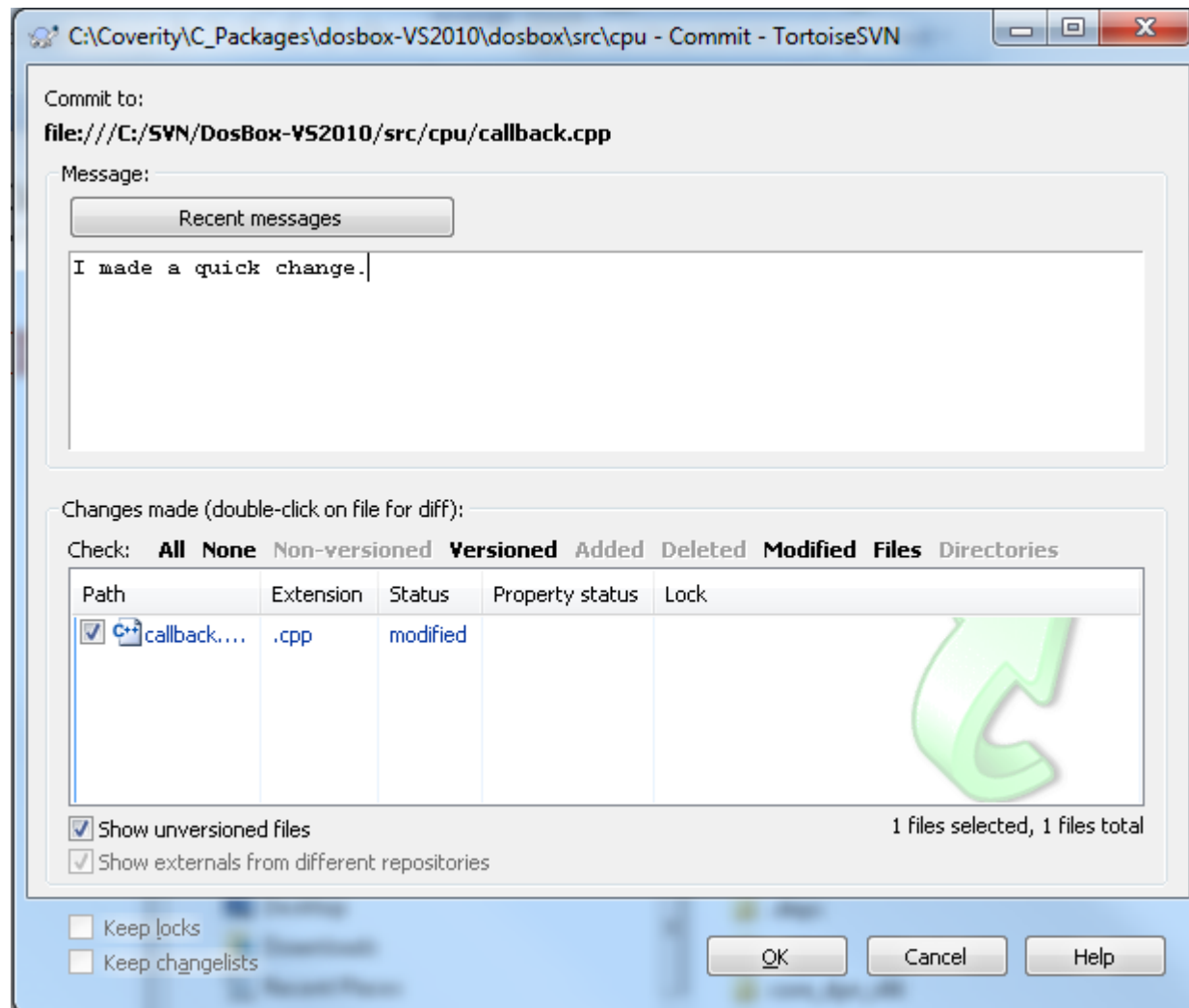
CallBack_Handler CallBack_Handlers[CB_MAX];
char* CallBack_Description[CB_MAX];

static Bitu call_stop,call_idle,call_default,call_default2;
Bitu call_priv_io;

34,0-1 4%
```

Continuous Integration Example

Jenkins Integration – agile execution



Continuous Integration Example

Jenkins Integration – agile execution

Jenkins

search



Jenkins » DosBox » #12

ENABLE AUTO REFRESH

[Back to Project](#)

[Status](#)

[Changes](#)

[Console Output](#)

[Edit Build Information](#)

[Polling Log](#)

[Tag this build](#)

[Coverity Defects](#)

[Previous Build](#)



Build #12 (Aug 4, 2011 2:22:27 PM)

Failed



Revision: 9
Changes

1. [\(detail\)](#)
2. [\(detail\)](#)



[Started by an SCM change](#)



[1](#) matching Coverity defect(s) found.

Delete this build

[add description](#)

Started 19 min ago

Took [1 min 22 sec](#)

A Continuous
Integration
build is
successful only
when it is free
of quality &
security defects

Summary

- Unified Approach to quality and security is possible with static analysis
- A(a)gile introduces some more requirements
- Address By:
 - Accurate analysis for quality & security on developers desktop
 - Analysis with every build
 - Analysis integration with agile tools and workflow



Questions?